Pattern Matching under Polynomial Transformation*

Ayelet Butman,† Peter Clifford,‡ Raphaël Clifford,§ Markus Jalsenius,§
Noa Lewenstein,¶ Benny Porat,‖ Ely Porat‖ and Benjamin Sach**

Abstract

We consider a class of pattern matching problems where a normalising transformation is applied at every alignment. Normalised pattern matching plays a key role in fields as diverse as image processing and musical information processing where application specific transformations are often applied to the input. By considering the class of polynomial transformations of the input, we provide fast algorithms and the first lower bounds for both new and old problems.

Given a pattern of length m and a longer text of length n where both are assumed to contain integer values only, we first show $O(n \log m)$ time algorithms for pattern matching under linear transformations even when wildcard symbols can occur in the input. We then show how to extend the technique to polynomial transformations of arbitrary degree. Next we consider the problem of finding the minimum Hamming distance under polynomial transformation. We show that, for any $\varepsilon > 0$, there cannot exist an $O(nm^{1-\varepsilon})$ time algorithm for additive and linear transformations conditional on the hardness of the classic 3Sum problem. Finally, we consider a version of the Hamming distance problem under additive transformations with a bound k on the maximum distance that need be reported. We give a deterministic $O(nk \log k)$ time solution which we then improve by careful use of randomisation to $O(n\sqrt{k \log k} \log n)$ time for sufficiently small k. Our randomised solution outputs the correct answer at every position with high probability.

1 Introduction

We consider pattern matching problems where the task is to find the distance between a pattern and every substring of the text of suitable length. In the class of problems we consider, the values in the pattern can first be transformed so as to minimise this

* The results in Section 2, except for those relating to higher degree polynomials, appeared in preliminary form in "Self-normalised Distance with Don't Cares", CPM '07. Section 4 contains revised and rewritten versions of results from "Jump-Matching with Errors", SPIRE '07.

† Department of Computer Science, Holon Institute of Technology, Holon, Israel.
‡ Department of Statistics, University of Oxford, U.K.
§ Department of Computer Science, University of Bristol, U.K.
¶ Department of Computer Science, Netanya Academic College, Israel.
‖ Department of Computer Science, Bar-Ilan University, Israel.
** Department of Computer Science, University of Warwick, U.K.

distance. Further, the selection of which transformation to apply, which is possibly distinct for each alignment, forms part of the problem that is to be solved. This class of problems generalises the well known problem of exact matching with wildcards [CH02, CC07] as well as the set of problems known previously as transposition invariant matching [MNU05], both of which come from the pattern matching literature. However as we will see, it is considerably broader than both with applications in both image processing and musical information retrieval.

By way of a first motivation for our work, consider a fundamental problem in image processing which is to measure the similarity between a small image segment or template and regions of comparable size within a larger scene. It is well known that the cross-correlation between the two can be computed efficiently at every position in the larger image using the fast Fourier transform (FFT). In practice, images may differ in a number of ways including being rotated, scaled or affected by noise. We consider here the case where the intensity or brightness of an image occurrence is unknown and where parts of either image contain *don't care* or *wildcard* pixels, i.e. pixels that are considered to be irrelevant as far as image similarity is concerned. As an example, a rectangular image segment may contain a facial image and the objective is to identify the face in a larger scene. However, some faces in the larger scene are in shadow and others in light. Furthermore, background pixels around the faces may be considered to be irrelevant for facial recognition and these should not affect the search algorithm.

In order to overcome the first difficulty of varying intensity within an image, a standard approach is to compute the *normalised* distance when comparing a template to part of a larger image. Thus both template and image are transformed or rescaled in order to make any matches found more meaningful and to allow comparisons between matches at different positions. Within the image processing literature the accepted method of normalisation is to scale the mean and variance of the template and image segments. We take a slightly different although related approach to normalisation which will allow to us to show a number of natural generalisations.

We start by defining measures of distance between a pattern P and text T, where P is a string of length m and T is a string of length $n \geqslant m$, both over the integers. The squared L_2 or Euclidean distance between the pattern and the text at position i is

$$\sum_{j=0}^{m-1} \big(P[j] - T[i+j]\big)^2 .$$

In this case, for each $i \in \{0, \ldots, n-m\}$, the pattern can be normalised, or fitted as closely as possible to the text, by transforming the input to minimise the distance.

In the case of degree one polynomial transformations, the normalised L_2 distance between the pattern and the text at position i can now be written as

$$\min_{\alpha,\beta} \sum_{j=0}^{m-1} \big(\alpha + \beta P[j] - T[i+j]\big)^2 ,$$

where the minimisation is over rational values of α and β. The minimisation is per

alignment of P and T, hence the values of α and β may (and probably will) differ between the positions i.

We also consider the case when the input alphabet is augmented with the special wildcard symbol, denoted "$\star$". A position where either the pattern or text has a wildcard will not contribute to the distance. That is, the minimisation is carried out using the sum of the remaining terms. Details are given in the problem definitions in the next section.

1.1 Problems and our results

The words *shift* and *scale* are used to refer to additive and multiplicative transformations of the pattern, respectively. The input to all our problems is a text T of length n and a pattern P of length m, and the output is a problem specific distance $d(i)$ between P and T at every position $i \in \{0, \ldots, n-m\}$ of the text. To avoid overloading variable names, we give the distance $d(i)$ a unique name for each problem.

Problem 1 (SHIFT-$L_2^\star$)**.** *Normalised L_2 distance under shifts. Wildcards are allowed.*

$$d_2^+(i) \stackrel{\text{def}}{=} \min_{\alpha} \sum_{j=0}^{m-1} \big(\alpha + P[j] - T[i+j]\big)^2 .$$

When either $P[j] = \star$ or $T[i+j] = \star$, the contribution of the pair to the sum $d_2^+(i)$ is taken to be zero. The minimisation is carried out using the sum of the remaining terms.

Next we define the normalised L_2 distance under shifts and scaling, corresponding to a degree one polynomial transformation of the values of the pattern.

Problem 2 (SHIFTSCALE-$L_2^\star$)**.** *Normalised L_2 distance under shifts and scaling. Wildcards are allowed.*

$$d_2^1(i) \stackrel{\text{def}}{=} \min_{\alpha,\beta} \sum_{j=0}^{m-1} \big(\alpha + \beta P[j] - T[i+j]\big)^2 .$$

When either $P[j] = \star$ or $T[i+j] = \star$, the contribution of the pair to the sum $d_2^1(i)$ is taken to be zero. The minimisation is carried out using the sum of the remaining terms.

We show that both SHIFT-$L_2^\star$ and SHIFTSCALE-$L_2^\star$ can be solved in $O(n \log m)$ time by the use of FFTs of integer vectors. Our results are stated in Theorems 10 and 11. We assume the RAM model of computation throughout in order to be consistent with previous work on matching with wildcards. Further, our techniques also provide $O(n \log m)$ time solutions (Theorems 12 and 13) to the problems of exact shift matching with wildcards (SHIFT-EXACT*) and exact shift-scale matching with wildcards (SHIFTSCALE-EXACT*), formally defined as follows.

Problem 3 (SHIFT-EXACT*)**.** *Normalised exact matching under shifts. Wildcards are allowed.*

$$d_{\mathrm{E}}^+(i) \stackrel{\text{def}}{=} \begin{cases} 1, & \exists\, \alpha \text{ st. } \alpha + P[j] = T[i+j] \text{ for all } j \in \{0, \ldots, m-1\}; \\ 0, & \text{otherwise.} \end{cases}$$

Every position j where either $P[j] = \star$ or $T[i+j] = \star$ is ignored.

Problem 4 (ShiftScale-Exact*)**.** *Normalised exact matching under shifts and scaling. Wildcards are allowed. The problem is defined similarly to* Shift-Exact**, only that we check whether there exist α and β st. $\alpha + \beta P[j] = T[i+j]$ for all positions j (except positions where $P[j] = \star$ or $T[i+j] = \star$).*

We will also discuss extensions to pattern transformations under polynomials of higher degree in Section 2. In terms of normalised L_2 distance we give the following definition.

Problem 5 (Poly-r-$L_2^\star$)**.** *Normalised L_2 distance under degree-r polynomial transformation. Wildcards are allowed. Let $f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_r x^r$ be a polynomial of degree r with $r \geqslant 1$.*

$$d_2^r(i) \stackrel{\text{def}}{=} \min_{\alpha_0,\ldots,\alpha_r} \sum_{j=0}^{m-1} \big(f(P[j]) - T[i+j]\big)^2 .$$

When either $P[j] = \star$ or $T[i+j] = \star$, the contribution of the pair to the sum $d_2^r(i)$ is taken to be zero. The minimisation is carried out using the sum of the remaining terms.

Note that the problem ShiftScale-$L_2^\star$ is the same problem as Poly-r-$L_2^\star$ with degree $r = 1$. We will show that Poly-r-$L_2^\star$ can be solved in $O(rn \log m + r^w n)$ time, where w is the exponent for matrix multiplication (e.g., $w \approx 2.38$ when using the Coppersmith-Winograd algorithm).

The second main topic of our work is on normalised pattern matching problems under the Hamming distance. The Hamming distance is perhaps the most commonly considered measure of distance between strings in the field of pattern matching. We therefore define related normalised versions of our pattern matching problems in a similar way to before.

Problem 6 (Shift-Ham)**.** *Normalised Hamming distance under shifts. Wildcards are not allowed.*

$$d_{\mathrm{H}}^+(i) \stackrel{\text{def}}{=} \min_{\alpha} \big|\, \{\, j \mid \alpha + P[j] \neq T[i+j] \,\} \,\big| .$$

Problem 7 (ShiftScale-Ham)**.** *Normalised Hamming distance under shifts and scaling. Wildcards are not allowed.*

$$d_{\mathrm{H}}^1(i) \stackrel{\text{def}}{=} \min_{\alpha,\beta} \big|\, \{\, j \mid \alpha + \beta P[j] \neq T[i+j] \,\} \,\big| .$$

Previously it has been shown that Shift-Ham, sometimes also referred to as transposition invariant matching, can be solved in $O(nm \log m)$ time [MNU05]. It has been tempting to believe that it might be possible to improve this time complexity, particularly as there exist algorithms for standard non-normalised pattern matching under the Hamming distance which take $O(n\sqrt{m \log m})$ time [Abr87, Kos87]. We show by reductions from the well known 3Sum problem that for both shift and shift-scale matching under the Hamming distance there cannot exist an $O(nm^{1-\varepsilon})$ time algorithm for any $\varepsilon > 0$ (Theorems 18 and 20).

To circumvent this new conjectured lower bound, we consider as our last problem a shift version of the k-mismatch problem. In the k-mismatch problem, the Hamming distance is to be reported at every alignment as long as it is at most k. If it is greater than k then the algorithm is only required to report that the Hamming distance is large. We define the problem as follows.

Problem 8 (SHIFT-k-MISMATCH)**.** *Normalised k-mismatch under shifts. Wildcards are not allowed.*

$$d^+_{\mathrm{M}}(i) \stackrel{\text{def}}{=} \min(d^+_{\mathrm{H}}(i),\, k+1)\,.$$

We first give a simple deterministic $O(nk \log k)$ time solution (Theorem 24). We then consider a decision version of the problem where we output only the locations i where $d^+_{\mathrm{H}}(i) \leqslant k$ but not the Hamming distance at those locations. The decision version is defined as follows.

Problem 9 (SHIFT-k-DECISION)**.** *Normalised k-mismatch decision problem under shifts. Wildcards are not allowed.*

$$d^+_{\mathrm{D}}(i) \stackrel{\text{def}}{=} \begin{cases} 0, & d^+_{\mathrm{H}}(i) \leqslant k; \\ 1, & \textit{otherwise.} \end{cases}$$

Using randomisation we show how to solve this problem in $O(cn\sqrt{k \log k} \log n)$ time for the case that $k < \sqrt{m/6}$ (Theorem 33). Here c is a constant that can be chosen arbitrarily to fine tune the error probability. Namely, our algorithm outputs the correct answer at every alignment with probability at least $1 - 1/n^c$. We therefore succeed in breaking our newly introduced running time barrier provided by the reduction from 3SUM for a limited range of values of k.

1.2 Related work

Combinatorial pattern matching has concerned itself mainly with strings of symbolic characters where the distance between individual characters is specified by some convention. For the k-mismatch problem, an $O(nk)$ time algorithm was given in 1986 that uses constant time lowest common ancestor queries on the suffix tree of the pattern and text in a technique that has subsequently come to be known as 'kangaroo hopping' [LV86]. Almost 20 years afterwards, the asymptotic running time was finally improved in [ALP04] to $O(n\sqrt{k \log k})$ time by a method based on filtering, the suffix tree (with kangaroo hopping) and FFTs. In 2002, a deterministic $O(n \log m)$ time solution for exact matching with wildcards was given by Cole and Hariharan [CH02] and further simplified in [CC07]. In the same paper by Cole and Hariharan, an $O\big(n \log(\max(m, N))\big)$ time algorithm for the exact shift matching problem we consider in Section 2 was presented. Here N is the largest value in the input. The approach we take to provide a simpler solution for this problem is similar in spirit to that of [CC07].

There has also been some work in recent years on fast algorithms for distance calculation and approximate matching between numerical strings. A number of different

metrics have been considered, with for example, $O(n\sqrt{m \log m})$ time solutions found for the L_1 distance [Ata01, CCI05, ALPU05] and less-than matching [AF95] problems and an $O(\delta n \log m)$ time algorithm for the δ-bounded version of the L_∞ norm first discussed in [CI04] and then improved in [CCI05, LP05].

The most closely related work to ours comes under the heading of transposition invariant matching [LU00]. The original motivation for this problem was within musical information retrieval where musical search is to be performed invariant of pitch level transposition. The transposition invariant distance between two equal lengthed strings A and B is defined to be $\min_\alpha d(A + \alpha, B)$, where $A + \alpha$ is the string obtained from A by adding α to every value and the distance d between strings can be variously defined. Algorithms for transposition invariant Hamming distance, longest common subsequence (LCS) and Levenshtein (edit) distance amongst others were given in [MNU05] whose time complexities are close to the known upper bounds without transposition. We show, in Section 3, lower bounds for the special case of transposition invariant Hamming distance, which we named Shift-Ham. Normalised pattern matching is also of central interest in the image processing literature where normalisation is typically performed by scaling the mean and standard deviation of the template and each suitably sized image segment to be 0 and 1, respectively. An asymptotically fast method for performing normalised cross-correlation for template matching, also using FFTs, was given in [Lew95]. The methods we give in Section 2 have some broad similarity to their approach only in the use of FFTs to provide fast solutions. Due to the differences in the definition of normalisation between our work and theirs, the solutions we give are otherwise quite distinct.

As a general class of problems, pattern matching under polynomial transformation is to the best of our knowledge new. However, if we allow the degree of the polynomial transformation to increase to m, then determining for which alignment the normalised distance equals zero is equivalent to the known problem of function matching. Function matching has a deterministic $O(n|\Sigma_P| \log m)$ time solution, where $|\Sigma_P|$ is the size of the pattern alphabet, and a faster randomised algorithm which runs in $O(n \log n)$ time and has failure probability $1/n$ [AALP06].

1.3 Basic notation

For a string X of length ℓ, we write $X[i]$ to denote the ith character of X such that $X = X[0]\,X[1]\,X[2] \cdots X[\ell - 1]$ (the first index is always zero). The s-length substring of X starting at position i is denoted $X[i \ldots i + s - 1]$. For two strings X and Y, the notion $X \| Y$ is used to denote the string formed by concatenating X and Y in that order. All strings in this paper are over the integer alphabet. Therefore, $X[j]Y[j]$ denotes the product of the numerical characters $X[j]$ and $Y[j]$. If strings X and Y are of equal length, we use the notation $X \cdot Y$ for the string with characters $(X \cdot Y)[i] = X[i]Y[i]$. This element-wise arithmetic is used similarly for addition, subtraction, division and power. For example, the ith symbol of X^2/Y is $X[i]^2/Y[i]$. For a real value k, the scalar multiplication kX is the string $(kX)[i] = kX[i]$.

The notation $\mathrm{Ham}(X, Y)$ will be used to denote the Hamming distance between equal

lengthed strings X and Y:

$$\mathrm{Ham}(X,Y) \stackrel{\text{def}}{=} \left|\{\, i \mid X[i] \neq Y[i] \,\}\right|.$$

Throughout this paper we use T to denote the text and P for the pattern. We use n to denote the length of T and m for the length of P.

Our algorithms in Section 2 make extensive use of FFTs. An important property of the FFT is that the cross-correlation, defined as

$$(T \otimes P)[i] \stackrel{\text{def}}{=} \sum_{j=0}^{m-1} P[j]T[i+j]\,,$$

can be calculated accurately and efficiently for all $i \in \{0,\ldots,n-m\}$ in $O(n \log m)$ time (see e.g. [CLR90], Chapter 32). The time complexity is reduced from $O(n \log n)$ to $O(n \log m)$ using a standard splitting trick which partitions the text into $2m$ length substrings which overlap each other by m characters. When it is clear from the context we use $\sum$ as an abbreviation for $\sum_{j=0}^{m-1}$.

We use "$\star$" for the single character wildcard symbol. Under arithmetics on strings, as defined above, we may think of a wildcard as having the value zero. This value is, however, inconsequential for our purposes, as all expressions in this paper have the property that whenever a wildcard symbol is involved in some arithmetics, it is multiplied by a zero.

We write $[n]$ to denote the set of integers $\{0 \ldots n-1\}$. We also say that $g(n) \in \tilde{\Omega}(h(n))$ if and only if $g(n) \in \Omega(h(n)/\log^c n)$ for some constant c, i.e $g(n) \in \Omega(h(n))$ up to log factors.

1.4 Organisation

The reminder of the paper is organised as follows. In Section 2 we discuss normalised pattern distance under L_2 distance (Shift-$L_2^\star$ and ShiftScale-$L_2^\star$) and the decision variants (Shift-Exact* and ShiftScale-Exact*). We also show how to extend the methods to transformations of higher degree polynomials (Poly-r-$L_2^\star$). Then in Section 3 we give running time lower bounds for Shift-Ham and ShiftScale-Ham by reduction from the 3Sum problem. In Section 4 we introduce our new deterministic and randomised algorithms for Shift-k-Mismatch and Shift-k-Decision. Finally, we conclude in Section 5 and set out some open problems.

2 Normalised L_2 distance

We give $O(n \log m)$ time solutions for shift and shift-scale versions of the normalised L_2 distance problem with wildcards. We further show that this enables us to solve the exact shift matching and exact shift-scale matching problems in the same time complexity for inputs containing wildcard symbols. Lastly we show how to extend our solutions to normalisation under polynomials of arbitrary degree.

Algorithm 1 Solution to SHIFT-$L_2^\star$.

1. Construct P' from P such that $P'[j] = 0$ if $P[j] = \star$, and $P'[j] = 1$ otherwise. Construct T' from T similarly.

2. Compute the following six cross-correlations:

$$\begin{array}{lll} C_1 = (T^2 \cdot T') \otimes P' & C_3 = T' \otimes (P^2 \cdot P') & C_5 = T' \otimes (P \cdot P') \\ C_2 = (T \cdot T') \otimes (P \cdot P') & C_4 = (T \cdot T') \otimes P' & C_6 = T' \otimes P' \end{array}$$

3. Return $A = C_1 - 2C_2 + C_3 - \big((C_4 - C_5)^2 / C_6\big)$. We have $d_2^+(i) = A[i]$. For positions i where $C_6[i] = 0$ we have $d_2^+(i) = 0$.

2.1 Normalised L_2 distance under shifts

In order to handle wildcards, we define two new strings P' and T' obtained from P and T, respectively, such that $P'[j] = 0$ if $P[j] = \star$, and $P'[j] = 1$ otherwise. Similarly, $T'[i] = 0$ if $T[i] = \star$, and $T'[i] = 1$ otherwise. We can now express the shift normalised L_2 distance at position i as

$$d_2^+(i) = \min_\alpha \sum_{j=0}^{m-1} \Big(\big(\alpha + P[j] - T[i+j]\big)^2 \cdot P'[j] \cdot T'[i+j] \Big) .$$

Algorithm 1 shows how to compute $d_2^+(i)$ for all positions i. Correctness and running time is given in the following theorem.

Theorem 10. *The shift version of the normalised L_2 distance with wildcards problem (SHIFT-$L_2^\star$) can be solved in $O(n \log m)$ time.*

Proof. Consider Algorithm 1. We first analyse the running time. Step 1 requires only single passes over the input. Similarly, $(P^2 \cdot P')$, $(P \cdot P')$, $(T \cdot T')$ and $(T^2 \cdot T')$ can all be calculated in linear time once T' and P' are known. Using the FFT, the six cross-correlations in Step 2 can be calculated in $O(n \log m)$ time. The final vector of Step 3 is obtained in linear time. Thus, $O(n \log m)$ is the overall time complexity of the algorithm.

To show correctness we consider the minimum value of

$$A[i] = \sum_{j=0}^{m-1} \Big(\big(\alpha + P[j] - T[i+j]\big)^2 \cdot P'[j] \cdot T'[i+j] \Big) . \tag{1}$$

This can be obtained by differentiating with respect to α and obtaining the minimising value. Solving

$$\frac{\partial A[i]}{\partial \alpha} = 2 \sum_{j=0}^{m-1} \Big(\big(\alpha + P[j] - T[i+j]\big) \cdot P'[j] \cdot T'[i+j] \Big) = 0$$

Algorithm 2 Solution to SHIFTSCALE-$L_2^\star$.

1. Construct P' from P such that $P'[j] = 0$ if $P[j] = \star$, and $P'[j] = 1$ otherwise. Construct T' from T similarly.

2. Compute the following six cross-correlations:

$$\begin{array}{lll} C_1 = (T^2 \cdot T') \otimes P' & C_3 = T' \otimes (P^2 \cdot P') & C_5 = T' \otimes (P \cdot P') \\ C_2 = (T \cdot T') \otimes (P \cdot P') & C_4 = (T \cdot T') \otimes P' & C_6 = T' \otimes P' \end{array}$$

3. Compute

$$B_1 = C_3 \cdot C_4 - C_2 \cdot C_5\,, \quad B_2 = C_3 \cdot C_6 - C_5^2\,, \quad B_3 = C_2 - \frac{C_4 \cdot C_5}{C_6}\,, \quad B_4 = C_3 - \frac{C_5^2}{C_6}$$

and compute $\widehat{\alpha} = B_1/B_2$ and $\widehat{\beta} = B_3/B_4$. At positions i where $C_6[i] = 0$, set $\widehat{\alpha}[i] = \widehat{\beta}[i] = 0$. At positions i where $B_2[i] = 0$ and $C_6[i] \neq 0$, set $\widehat{\alpha}[i] = C_4/C_6$ and $\widehat{\beta}[i] = 0$.

4. Return $B = (\widehat{\alpha}^2 \cdot C_6) + 2(\widehat{\alpha} \cdot \widehat{\beta} \cdot C_5) - 2(\widehat{\alpha} \cdot C_4) + (\widehat{\beta}^2 \cdot C_3) - 2(\widehat{\beta} \cdot C_2) + C_1$.
We have $d_2^1(i) = B[i]$.

gives us the value

$$\widehat{\alpha} = \frac{\sum \Big(\big(T[i+j] - P[j]\big) \cdot P'[j] \cdot T'[i+j] \Big)}{\sum P'[j] \cdot T'[i+j]} = \frac{\big((T \cdot T') \otimes P'\big) - \big((P \cdot P') \otimes T'\big)}{P' \otimes T'}\,,$$

where $\widehat{\alpha}[i]$ is the minimising value at position i. Substituting $\alpha = \widehat{\alpha}$ into Equation (1), expanding and collecting terms, we obtain the final answer as

$$A = C_1 - 2C_2 + C_3 - \frac{(C_4 - C_5)^2}{C_6}\,,$$

where $C_1, \ldots, C_6$ are the correlations defined in Algorithm 1.

Lastly we observe that when $C_6[i] = (T' \otimes P')[i] = 0$ there is a wildcard at every position in the alignment of P and T. Here the shift normalised L_2 distance is defined to be 0. □

2.2 Normalised L_2 distance under shift-scale

Similarly to the shift version of normalised L_2 distance in the previous section, we can now solve the shift-scale version. The solution is slightly more involved but the running time remains the same. Algorithm 2 sets out the main steps to achieve this and the result is summarised in the following theorem.

Theorem 11. *The shift-scale version of the normalised L_2 distance with wildcards problem (SHIFTSCALE-$L_2^\star$) can be solved in $O(n \log m)$ time.*

Proof. Consider Algorithm 2. Notice that the same six correlations as in Algorithm 1 have to be calculated. The additional strings in Step 3 require linear time, as well as producing the output in Step 4. Hence the overall running time is $O(n \log m)$.

Similarly to Equation (1) we can express the shift-scale version of the normalised L_2 distance at position i as

$$B[i] \;=\; \sum_{j=0}^{m-1} \Big(\big(\alpha + \beta P[j] - T[i+j]\big)^2 \cdot P'[j] \cdot T'[i+j] \Big) \,. \tag{2}$$

By minimising this expression with respect to both α and β we get a system of two simultaneous linear equations

$$\begin{aligned} \frac{\partial B[i]}{\partial \alpha} \;&=\; 2 \sum_{j=0}^{m-1} \Big(\big(\alpha + \beta P[j] - T[i+j]\big) \cdot P'[j] \cdot T'[i+j] \Big) \;=\; 0 \,, \\ \frac{\partial B[i]}{\partial \beta} \;&=\; 2 \sum_{j=0}^{m-1} \Big(\big(\alpha + \beta P[j] - T[i+j]\big) \cdot P[j] \cdot P'[j] \cdot T'[i+j] \Big) \;=\; 0 \,. \end{aligned}$$

By solving this system and using the definitions of $B_1, \ldots, B_4$ in Algorithm 2, we get the minimising values

$$\widehat{\alpha} = \frac{B_1}{B_2} \qquad \text{and} \qquad \widehat{\beta} = \frac{B_3}{B_4} \,.$$

For some positions i, the solution to the system might not be unique. This happens at alignments i for which every position $i+j$ has a wildcard, hence $C_6[i] = 0$. Here we avoid illegal division by zero by simply setting both $\widehat{\alpha}[i]$ and $\widehat{\beta}[i]$ to zero (any value would do). A non-unique solution also occurs at alignments i where all $P[j]$ are identical over every non-wildcard position $i+j$. This is characterised by $B_2[i] = 0$. To see this, observe that $C_5[i]^2 \leqslant C_3[i] C_6[i]$ by Cauchy-Schwarz inequality. Here we set (arbitrarily) $\widehat{\beta}[i] = 0$ and therefore obtain the minimising value $\widehat{\alpha}[i] = C_4 / C_6$.

At Stage 4, $\widehat{\alpha}$ and $\widehat{\beta}$ contain the minimising values for α and β at every position. We substitute these into Equation (2) and expand. This gives us the expression for B. □

2.3 Exact shift and shift-scale matching with wildcards

For the exact shift matching problem with wildcards, SHIFT-EXACT*, a match is said to occur at location i if, for some shift α and for every position j in the pattern, either $\alpha + P[j] = T[i+j]$ or at least one of $P[j]$ and $T[i+j]$ is the wildcard symbol. Cole and Hariharan [CH02] introduced a new coding for this problem that maps the string elements into 0 for wildcards and complex numbers of modulus 1 otherwise. The FFT is then used to find the (complex) cross-correlation between these coded strings, and finally a shift match is declared at location i if the ith element of the modulus of the cross-correlation is equal to $(P' \otimes T')[i]$.

Our Algorithm 1 provides a straightforward alternative method for shift matching with wildcards. It has the advantage of only using simple integer codings. Since Algorithm 1 finds the minimum L_2 distance at location i, over all possible shifts, it is only

necessary to test whether this distance is zero. The running time for the test is then $O(n \log m)$ since it is determined by the running time of Algorithm 1.

Theorem 12. *The problem of exact shift matching with wildcards (*Shift-Exact**) can be solved in $O(n \log m)$ time.*

The exact shift-scale matching problem with wildcards, ShiftScale-Exact*, can be solved similarly by applying Algorithm 2.

Theorem 13. *The problem of exact shift-scale matching with wildcards (*ShiftScale-Exact**) can be solved in $O(n \log m)$ time.*

2.4 Normalised L_2 distance under higher degree transformations

We can now consider the problem of computing the normalised L_2 distance under general polynomial transformations. The problem, which we termed Poly-r-$L_2^\star$, was defined in Problem 5. Recall that we let

$$f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_r x^r$$

be a polynomial of degree $r \geqslant 1$. Similarly to the shift and shift-scale versions of the normalised L_2 distance we consider the minimum value of

$$D[i] = \sum_{j=0}^{m-1} \Big(\big(f(P[j]) - T[i+j] \big)^2 \cdot P'[j] \cdot T'[i+j] \Big) \,. \tag{3}$$

By differentiating with respect to each α_k in turn, giving

$$\frac{\partial D[i]}{\partial \alpha_k} \;=\; 2 \sum_{j=0}^{m-1} \Big(\big(f(P[j]) - T[i+j] \big) \cdot P[j]^k \cdot P'[j] \cdot T'[i+j] \Big) \;=\; 0 \,,$$

we obtain a system of $r+1$ linear equations in $r+1$ unknowns for each alignment i of the pattern and text. We need to solve these equations and then substitute the minimising α_k values back into Equation (3) as we did in the proof of Theorem 11. This procedure is captured by the following theorem.

Theorem 14. *The normalised L_2 distance problem with wildcards under polynomial transformations of degree r (*Poly-r-$L_2^\star$*) can be solved in $O(rn \log m + r^{2.38} n)$ time.*

Proof. To compute the coefficients for the first linear equation for α_0 we need to perform $O(r)$ cross-correlations. However, for each subsequent equation for $\alpha_1 \ldots \alpha_r$ we only need to perform a constant number of new cross-correlations. Therefore the total number of cross-correlations is $O(r)$ to give the coefficients of all the equations, taking $O(rn \log m)$ time overall. The time to solve the systems of $O(r)$ equations in $O(r)$ unknowns is $O(r^w)$ per alignment i, where w is the exponent for matrix multiplication. This gives $O(nr^w)$ time or $O(nr^{2.38})$ using the algorithm of Coppersmith and Winograd [CW90].

Once the equations have been solved, and the minimising values of α_k calculated, they are then substituted into the expression for D in Equation (3). To calculate the final values $D[i]$ we require $O(r)$ cross-correlations to be computed as well as $O(r^2)$ products of vectors of length m. The overall time complexity is therefore $O(rn \log m + r^{2.38}n + r^2 m)$. □

This method is of particular relevance for low degree polynomials, or at least polynomials whose degree is less than the number of distinct values in the pattern. However, if the degree r is greater than the number of distinct values in the pattern, then there exists a suitable polynomial f for any mapping we should choose. This gives us a straightforward $O(nm)$ time solution by considering each position of the pattern in the text independently and ignoring any values aligned with wildcards in either the pattern or text. For each such position we need only set $f(P[j])$ to be the mean of the values in the text that align with a value equal to $P[j]$ in the pattern.

3 Lower bounds for Hamming distance

In this section we will show that no $O(nm^{1-\varepsilon})$ time algorithm can exist for neither Shift-Ham or ShiftScale-Ham conditional on the hardness of the classic 3Sum problem. One formulation of the 3Sum problem is given below.

Definition 15 (3Sum)**.** Given a set of s positive integers, determine whether there are three elements a, b, c in the set such that $a + b = c$.

The 3Sum problem can be solved in $O(s^2)$ time and it is a long standing conjecture that this is essentially the best possible. The problem has been extensively discussed in the literature, where Gajentaan and Overmars [GO95] were the first to introduce the concept of 3Sum-hardness (see definition below) to show that a wide range of problems in computational geometry are at least as hard as the 3Sum problem. One example is the GeomBase problem, defined below, which we will use in one of our reductions in this section. See [Kin04] for a survey of problems from computational geometry whose hardness relies on that of 3Sum.

Definition 16 (GeomBase)**.** Given a set of s points with integer coordinates on three horizontal lines $y = 0$, $y = 1$ and $y = 2$, determine whether there exists a non-horizontal line containing three of the points.

Although an $\tilde{\Omega}(s^2)$ lower bound for 3Sum is only conjectured, it has been shown that under certain restricted models of computation, $\Omega(s^2)$ is a true lower bound (see [ES95, Eri99a, Eri99b]). Under models that allow more direct manipulation of numbers instead of just real arithmetic, such as the word-RAM model, an almost $\log^2 s$ factor improvement to the standard $O(s^2)$ solution has been shown to be possible under the Las Vegas model of randomisation (see [BDP05]). Nevertheless, a 3Sum-hardness result for a problem is a strong indication that finding an $O(s^{2-\varepsilon})$ time solution is going to be a challenging task.

Before we show that Shift-Ham and ShiftScale-Ham are both 3Sum-hard, we provide a brief but formal discussion about reductions and define 3Sum-hardness.

3.1 3SUM reductions

Following the definitions of [GO95] where 3SUM-hardness was first introduced, we say that a problem A is $g(s)$-*solvable using a problem* B if and only if every instance of A of size s can be solved using a constant number of instances of B of at most $O(s)$ size and $O(g(s))$ additional time. We denote this as $A \lll_{g(s)} B$. When $g(s)$ is sufficiently small, lower bounds for A carry over to B. A problem B is 3SUM-*hard* if 3SUM $\lll_{g(s)} B$ and $g(s) = o(s^{2-\varepsilon})$ for some constant $\varepsilon > 0$. In the definition of 3SUM-hardness of [GO95], the requirement was that $g(s) = o(s^2)$, however, to scale with more powerful models of computation, we require that $g(s) = o(s^{2-\varepsilon})$. If $A \lll_{g(s)} B$ and $B \lll_{g(s)} A$ then we say that A and B are $g(s)$-*equivalent*.

In the following section we will show that 3SUM $\lll_{s \log s}$ SHIFT-HAM where the instance size of SHIFT-HAM is a text of length $n = 5s$ and a pattern of length $m = 3s$.

In the literature there are a variety of definitions of the 3SUM problem. They differ only slightly in their formulations and are all equivalent. One common definition, used as the "base problem" in [GO95], is formulated as follows. Given a set of s integers, determine whether there are three elements a, b, c in the set such that $a + b + c = 0$. Without too much work, one can show that this definition is $O(s)$-equivalent with Definition 15 of 3SUM above (small modifications of the proof of Theorem 3.1 in [GO95] can be used to prove this). Further, it was shown in [GO95] that GEOMBASE is $O(s)$-equivalent to 3SUM.

3.2 3SUM-hardness of SHIFT-HAM

In this section we show that SHIFT-HAM is 3SUM-hard.

Lemma 17. 3SUM $\lll_{s \log s}$ SHIFT-HAM *where the instance size of* SHIFT-HAM *is a text of length* $5s$ *and a pattern of length* $3s$.

Proof. Let the set S be an instance of 3SUM of size $s = |S|$. First we sort all elements of S so that $S = \{x_1, \ldots, x_s\}$ where $x_1 < x_2 < \cdots < x_s$. Let $y_1 = 2x_s + 1$ and for $i \in \{2, \ldots, 2s\}$, let $y_i = y_{i-1} + 1$. Thus, $x_s < y_1 < \cdots < y_{2s}$. We define the following s-length strings over the alphabet $\{x_1, \ldots, x_s\} \cup \{y_1, \ldots, y_{2s}\} \cup \{0\}$.

$$S_0 = 0\,0 \cdots 0 \quad (s \text{ zeros}) \qquad S_3 = y_{s+1}\, y_{s+2} \cdots y_{2s}$$
$$S_1 = x_1\, x_2 \cdots x_s \qquad S_4 = x_s\, x_{s-1} \cdots x_1$$
$$S_2 = y_1\, y_2 \cdots y_s$$

We now construct an instance of SHIFT-HAM specified by

$$T = S_0 \| S_1 \| S_2 \| S_1 \| S_3 \qquad \text{and} \qquad P = S_4 \| S_0 \| S_0\,.$$

The text T has length $n = 5s$ and the pattern P has length $m = 3s$. First we show that if there are elements $a, b, c \in S$ such that $a + b = c$ then there is a position i such that the shift-normalised Hamming distance between P and $T[i \ldots i + m - 1]$ is at most $m - 2$. We will then show that if no such three elements exist then the shift-normalised

Hamming distance between P and every m-length substring of T is strictly greater than $m-2$.

As an illustrative example, suppose that S contains seven elements and suppose that $x_4 + x_3 = x_6$. Consider the alignment of P and T where x_4 in P is aligned with x_6 in T:

T: $0\ 0\ 0\ 0\ 0\ 0\ 0\ x_1\ x_2\ x_3\ x_4\ x_5\ \boxed{x_6}\ x_7\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7\ x_1\ x_2\ \boxed{x_3}\ x_4\ x_5\ x_6\ x_7\ y_8\ y_9\ y_{10}\ y_{11}\ y_{12}\ y_{13}\ y_{14}$

P: $x_7\ x_6\ x_5\ \boxed{x_4}\ x_3\ x_2\ x_1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \boxed{0}\ 0\ 0\ 0\ 0\ 0\ 0$

We observe that shifting the pattern by x_3 will induce two matches, marked with the squares above. Thus, the shift-normalised Hamming distance is at most $m-2$ (in fact, it is exactly $m-2$). It should be easy to see how this generalises to any size of S and any three elements $a, b, c \in S$ such that $a + b = c$. Namely, the alignment in which a is aligned with c has Hamming distance at most $m-2$ since there must also be a match at the position where 0 aligned with b. The construction of P and T ensures that there is always an alignment that captures these matches.

Now suppose there are no elements $a, b, c \in S$ such that $a + b = c$. Consider a fixed alignment of P and T. We will show that there can be at most one match under any shift. By construction of P and T, the zeros in P are all aligned with distinct symbols in T. Hence for any shift, at most one of these zeros can be involved in a match. The non-zero symbols of P (i.e., the s-length prefix of P) appear in strictly decreasing order and are aligned with an s-length substring of T whose elements appear in non-decreasing order. Therefore, under any shift, at most one of the non-zero symbols in P can be involved in a match. It remains to show that there is no shift such that both a zero and a non-zero symbol in P are simultaneously involved in a match. First, we observe that if there is a match between a 0 in P and some y_j in T then there can be no other match as every non-zero symbol in P is aligned with a value that is less than y_j. Suppose therefore that there is a match between a 0 in P and some x_j in T (i.e., the shift is x_j). We need to consider three possible cases: there is also a match that involves some x_k in P aligned with either (i) a 0 in T, (ii) some y_ℓ in T or (iii) some x_ℓ in T. In case (i) the shift must be negative, hence is not compatible with the shift x_j. In case (ii) we can see that the shift must be greater that x_s (the largest elements in the set S), hence is not compatible with the shift x_j. In case (iii) we have that $x_k + x_j = x_\ell$, which contradicts the assumption that there are no elements $a, b, c \in S$ such that $a + b = c$. Thus, the shift-normalised Hamming distance is at least $m-1$ for any alignment of P and T.

Finally, we observe that the most time consuming part of the reduction is the sorting of S which could take $O(s \log s)$ time. This concludes the proof. □

Theorem 18. Shift-Ham *has no* $O(nm^{1-\varepsilon})$ *time algorithm, for any* $\varepsilon > 0$, *conditional on the hardness of the* 3Sum *problem.*

Proof. Given a 3Sum instance of size s, by Lemma 17 we construct a Shift-Ham instance of size $n = 5s$ and $m = 3s$ in $O(s \log s)$ time. If Shift-Ham has an $O(nm^{1-\varepsilon})$ time algorithm then 3Sum can be solved in $O(s^{2-\varepsilon})$ time. □

Notice that Shift-Ham has an $O(nm \log m)$ time solution [MNU05]. See Section 4.1 for details.

3.3 3Sum-hardness of ShiftScale-Ham

In this section we show that ShiftScale-Ham is 3Sum-hard.

Lemma 19. 3Sum $\ll_s$ ShiftScale-Ham *where the instance size of* Shift-Ham *is a pattern and text of length s each.*

Proof. We reduce from the GeomBase problem which is $O(s)$-equivalent to 3Sum. Before we describe the reduction we adopt a formulation of the GeomBase problem that differs slightly in notation. Instead of insisting on the points being on the horizontal lines $y = 0$, $y = 1$ and $y = 2$, we assume that the points are on the vertical lines $x = 0$, $x = 1$ and $x = 2$ and we want to determine whether there is a (non-vertical) line containing three points. Under this formulation, let S be an instance of GeomBase that contains the integer points $(x_1, y_1), (x_2, y_2), \ldots, (x_s, y_s)$, where every $x_j \in \{0, 1, 2\}$.

We construct an instance of ShiftScale-Ham that is specified by the text $T = y_1\, y_2 \cdots y_s$ and the pattern $P = x_1\, x_2 \cdots x_s$, both of length s. It should now be clear that ShiftScale-Ham returns the shift-and-scale normalised Hamming distance $s - 3$ (for the only alignment of P and T) if and only if there are two values α and β such that $\beta x_j + \alpha = y_j$ for three distinct positions j, which is equivalent to fitting a line through three points. Note that we minimise α and β over the rationals, and any line going through three points is indeed specified by rational values of α and β. Since the reduction takes linear time, we have proved the lemma. □

Theorem 20. ShiftScale-Ham *has no $O(nm^{1-\varepsilon})$ algorithm, for any $\varepsilon > 0$, conditional on the hardness of the* 3Sum *problem.*

Proof. Given a 3Sum instance of size s, by Lemma 19 we construct a ShiftScale-Ham instance of size $n = m = s$ in $O(s)$ time. If ShiftScale-Ham has an $O(nm^{1-\varepsilon})$ algorithm then 3Sum can be solved in $O(s^{2-\varepsilon})$ time. □

4 Normalised k-mismatch under shifts

In this section we consider two versions of the normalised k-mismatch problem under shifts, defined as Problems 8 and 9 in the introduction. Both problems are parameterised by an integer k. In the first problem, Shift-k-Mismatch, the output is the shift-normalised Hamming distance between P and T at every position for which the distance is k or less. Where the distance is larger than k, only $k+1$ is outputted. Recall from the introduction that the shift-normalised Hamming distance between P and $T[i \ldots i+m-1]$ is denoted $d_{\mathrm{H}}^{+}(i)$ and defined by

$$d_{\mathrm{H}}^{+}(i) \stackrel{\text{def}}{=} \min_{\alpha} \left| \{\, j \mid \alpha + P[j] \neq T[i+j] \,\} \right|.$$

In Section 4.2 we give a deterministic algorithm that solves Shift-k-Mismatch in $O(nk \log k)$ time.

In Section 4.3 we consider the the second version of shift-normalised k-mismatch, Shift-k-Decision, which unlike the previous problem only indicates with yes or no

whether the shift-normalised Hamming distance is k or less. We give a randomised solution to this decision problem with the improved running time $O(cn\sqrt{k \log k} \log n)$. The parameter c is a constant that can be chosen arbitrarily to fine tune the error probability. Namely, the probability that our algorithm outputs the correct answer at every alignment is at least $(1 - 1/n^c)$. The errors are one-sided such the algorithm will never miss reporting an alignment for which the shift-normalised Hamming distance is indeed k or less. Our algorithm requires that $k < \sqrt{m/6}$, hence it is suited to situations where the locations of text substrings similar to the pattern are required but the distances themselves are not needed.

4.1 The unbounded case

In [MNU05], Mäkinen, Navarro and Ukkonen gave an $O(nm \log m)$ time algorithm for the shift-normalised Hamming distance problem, SHIFT-HAM, which by definition solves the bounded, k-mismatch variant in $O(nm \log m)$ time also. We briefly recap their method by way of an introduction. First observe that the maximum number of matches for any alignment is exactly

$$m - d_{\mathrm{H}}^{+}(i) = \max_{\alpha} \{\, j \mid T[i+j] - P[j] = \alpha \,\} \,.$$

For each alignment i, this value can be obtained by creating an m-length array A_i, which we refer to as the *shift array*, defined by

$$A_i[j] = T[i+j] - P[j] \tag{4}$$

for all $j \in [m]$. This shift array is then sorted to find the most frequent value, which is the α that minimises $d_{\mathrm{H}}^{+}(i)$. The number of times it occurs is $m - d_{\mathrm{H}}^{+}(i)$. Computing this requires $O(m \log m)$ time per alignment and hence $O(nm \log m)$ time overall. In the next section we will reconsider A_i and demonstrate that it can be run-length encoded in $O(k)$ runs whenever $d_{\mathrm{H}}^{+} \leqslant k$.

4.2 A deterministic solution

The deterministic algorithm makes use of the notion of difference strings which were introduced in [LU00] and are defined as follows.

Definition 21. Let S be a string of length s. The *difference string* of S, denoted S_δ, is defined by

$$S_\delta[j] = S[j+1] - S[j]$$

for all $j \in [s-1]$. The length of S_δ is $s - 1$.

We will also make use of a generalisation of the difference string when we present our randomised algorithm in Section 4.3. The core of our deterministic shift-normalised k-mismatch algorithm is the relationship between the number of mismatches between P_δ and $T_\delta[i \ldots i+m-2]$ and the value of $d_{\mathrm{H}}^{+}(i)$. We begin in Lemma 22 below by showing

that if $d_{\mathrm{H}}^{+}(i)$ is small then the number of mismatches between the difference strings P_δ and T_δ is also small. In [MNU05] a related result was used to reduce the shift-normalised exact matching problem to the conventional exact matching problem. Specifically, they observed that in the special case that $k = 0$, the implication becomes an equivalence, i.e., $d_{\mathrm{H}}^{+}(i) = 0$ if and only if $P_\delta = T_\delta[i \dots i + m - 2]$. Unfortunately, this is not the case in general.

Lemma 22. *Let P be a pattern and T a text. For all i,*

$$d_{\mathrm{H}}^{+}(i) \leqslant k \implies \mathrm{Ham}\big(P_\delta, T_\delta[i \dots (i + m - 2)]\big) \leqslant 2k\,.$$

Proof. Let i be such that $d_{\mathrm{H}}^{+}(i) \leqslant k$ and therefore there exists an α such that for at most k distinct position $j \in [m]$ we have that $P[j] + \alpha \neq T[i + j]$. Further, at most $2k$ distinct positions $j \in [m - 1]$ have either $P[j] + \alpha \neq T[i + j]$ or $P[j + 1] + \alpha \neq T[i + j + 1]$. This implies that there are at least $(m - 1) - 2k$ distinct positions $j \in [m - 1]$ such that $P[j] + \alpha = T[i + j]$ and $P[j + 1] + \alpha = T[i + j + 1]$. By rearranging these equations, for any such j we have that $P[j + 1] - P[j] = T[i + j + 1] - T[i + j]$ and hence by Definition 21, $P_\delta[j] = T_\delta[1 + j]$. As required there are at most $2k$ mismatches (recall $|P_\delta| = m - 1$). □

Lemma 22 suggests the following strategy. First we find the leftmost up to $2k + 1$ mismatches between P_δ and $T_\delta[i \dots i + m - 2]$ at each alignment i. By Lemma 22 we can disregard any alignments with more than $2k$ mismatches. Finally we use the locations of these mismatches to infer $d_{\mathrm{H}}^{+}(i)$ at the remaining alignments.

The first step can be done using any k-mismatch (strictly $2k$-mismatch) algorithm which returns the locations of the mismatches. The well-known 'kangaroo' method of [LV86] achieves this in optimal $O(nk)$ time. The method is so named as it uses longest common extensions to 'hop' between mismatches in constant time. The discarding phase is trivial and therefore we only focus on computing $d_{\mathrm{H}}^{+}(i)$ from the locations of the (at most $2k$) mismatches between P_δ and $T_\delta[i \dots (i + m - 2)]$, where i is an arbitrary non-discarded alignment.

Recall from Section 4.1 the definition of the shift array A_i in Equation (4), and recall that the value of $m - d_{\mathrm{H}}^{+}(i)$ is the number of occurrences of the most frequent entry in A_i. We will now use the locations of the mismatches between P_δ and $T_\delta[i \dots i + m - 2]$ to obtain a run-length encoded version of A_i containing $O(k)$ runs. The key property we require is given in Lemma 23 which states that a matching substring in P_δ and $T_\delta[i \dots i + m - 2]$ corresponds to a run (a substring of equal values) in A_i. This immediately implies that A_i can be decomposed into at most $4k + 1$ runs. Specifically, one run of length 1 for each mismatch and an additional run for each stretch between mismatches.

Lemma 23. *If $P_\delta[\ell \dots r] = T_\delta[(i + \ell) \dots (i + r)]$ then $A_i[j] = A_i[\ell]$ for all $j \in \{\ell, \dots, r\}$.*

Proof. Suppose that $P_\delta[\ell \dots r] = T_\delta[(i + \ell) \dots (i + r)]$. We proceed by induction on $j \in \{\ell, \dots, r\}$. The base case $j = \ell$ is tautologically true. For the inductive step, let $j \in \{\ell + 1, \dots, r\}$. By the inductive hypothesis, we have that $A_i[j - 1] = T[i + j - 1] - P[j - 1] = A_i[\ell]$. As $P_\delta[j - 1] = T_\delta[i + j - 1]$, by Definition 21 (and rearranging the equation), we have $A_i[j] = T[i + j] - P[j] = T[i + j - 1] - P[j - 1] = A_i[\ell]$. □

Algorithm 3 Overview of deterministic solution to SHIFT-k-MISMATCH.

1. Compute the difference strings P_δ and T_δ by scanning P and T.
2. Run a $2k$-mismatch algorithm on P_δ and T_δ in order to find all alignments where the number of mismatches is at most $2k$. The $2k$-mismatch algorithm must also return the locations of the mismatches at any alignment where there are at most $2k$ mismatches.
3. Discard all alignments with more than $2k$ mismatches.
4. For each undiscarded alignment i, decompose A_i into at most $4k+1$ runs (substrings with a common value). The start and end points of the runs are determined by scanning the locations of the mismatches between P_δ and $T_\delta[i \ldots i+m-1]$.
5. Sort the runs in A_i by value in order to find the most frequent entry α in A_i. Then output $m - \left|\{\, j \mid A_i[j] = \alpha \,\}\right|$, which is the value $d_{\mathrm{H}}^{+}(i)$.

In Section 4.1 we discussed that the value of $d_{\mathrm{H}}^{+}(i)$ equals $m - \max_\alpha \left|\{\, j \mid A_i[j] = \alpha \,\}\right|$, which could be found by sorting and scanning A_i in $O(m \log m)$ time. However, we now have A_i in run-length encoded form (with $O(k)$ runs), therefore the time taken to find $d_{\mathrm{H}}^{+}(i)$ is reduced to $O(k \log k)$. Over all alignments, this gives $O(nk \log k)$ time as desired.

We can now give an overview of our deterministic algorithm for SHIFT-k-MISMATCH. The steps are described in Algorithm 3 and the overall running time is given in Theorem 24 below.

Theorem 24. *The shift-normalised k-mismatch problem (SHIFT-k-MISMATCH) can be solved deterministically in $O(nk \log k)$ time.*

Proof. The solution is outlined in Algorithm 3. Correctness follows directly from the discussion in this section. The time complexity of the five steps is as follows. By inspection of the definition, the difference strings computed in Step 1 require $O(n)$ time. Step 2 uses a $2k$-mismatch algorithm as a black box and can be performed in $O(nk)$ time by using for example the algorithm in [LV86]. Step 3 makes a single pass of the output of the $2k$-mismatch algorithm in $O(n)$ time. Step 4 constructs a run length encoded version of A_i for each undiscarded i. This requires scanning the $O(k)$ mismatches at each undiscarded alignment. Therefore Step 4 takes $O(nk)$ time. Step 5 scans and sorts each A_i which takes $O(k \log k)$ time per alignment as A_i is encoded by $O(k)$ runs. Overall the algorithm requires $O(nk \log k)$ time as claimed. □

4.3 An improved, randomised solution

We now present an improved solution to the shift-normalised k-mismatch problem which runs in $O(cn\sqrt{k \log k} \log n)$ time. The improved algorithm is for the case that $k < \sqrt{m/6}$ and is randomised. The errors are one-sided (false-positives) and it outputs the correct answer at all alignments with probability at least $1 - 1/n^c$ for any constant c. For each

position i, the algorithm gives a yes/no answer to the question "is $d_{\mathrm{H}}^{+}(i) \leqslant k$?". The algorithm does not output the actual distance $d_{\mathrm{H}}^{+}(i)$. Throughout this section, we use T_i as shorthand for $T[i \ldots i+m-1]$.

In Section 4.2 our deterministic algorithm made use of the locations of mismatches in the difference strings P_δ and $T_\delta[i \ldots i+m-1]$. Recall that the difference string S_δ was defined to give the differences between consecutive positions in a string S. That is, $S_\delta[j] = S[j+1] - S[j]$ for all j. A key observation was that $P_\delta[j] = T_\delta[i+j]$ if and only if $P[j] - T[i+j] = P[j+1] - T[i+j+1] = -\alpha$, i.e., the positions of $P[j]$ and $P[j+1]$ require the same shift α to match. However, there is no reason to consider only consecutive differences. In fact, as we will see, one may consider differences under any arbitrary permutation of the position set. This notion is formalised as follows.

Definition 25. Let S be a string of length s and $\pi : [m] \to [m]$ be a permutation. The *permuted difference string* of S under π, denoted S_π, is defined by

$$S_\pi[j] = S[\pi(j)] - S[j]$$

for all $j \in [s]$. The length of S_π is s.

Note that the permuted difference string S_π has length $|S|$ in contrast to the difference string S_δ of Definition 21 which has length $|S| - 1$.

The central idea of our improved algorithm is to use the value of $\mathrm{Ham}(P_\pi, (T_i)_\pi)$ to directly determine whether $d_{\mathrm{H}}^{+}(i) \leqslant k$ at each alignment i. In Definition 26 below we introduce the notion of a permutation being *k-tight* for some P, T_i. Intuitively, π is k-tight for P, T_i if we can infer directly from $\mathrm{Ham}(P_\pi, (T_i)_\pi)$ whether $d_{\mathrm{H}}^{+}(i) \leqslant k$.

Definition 26. Let π be a permutation, P a pattern and T_i a text substring. We say that π is *k-tight* for P, T_i if

$$d_{\mathrm{H}}^{+}(i) \leqslant k \iff \mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k\,.$$

It would of course be highly desirable to find a permutation π which is k-tight for all P, T_i and any k. However, we will see that this is in general not possible.

We begin by showing that any π has the property that $d_{\mathrm{H}}^{+}(i) \leqslant k$ implies that $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k$ for all P, T_i. To do so we first prove a general lemma which will also be useful later. Lemma 28 then gives the desired property and is a generalisation of Lemma 22 to arbitrary permutations.

Lemma 27. *Let π be a permutation, P a pattern and T_i a text substring. For all $j \in [m]$,*

$$P[j] - T_i[j] = P[\pi(j)] - T_i[\pi(j)] \iff P_\pi[j] = (T_i)_\pi[j]\,.$$

Proof. The left-hand side of the arrow is the same as $P[\pi(j)] - P[j] = T_i[\pi(j)] - T_i[j]$, which by Definition 25 is equivalent to the right-hand side of the arrow. □

Lemma 28. *Let π be a permutation, P a pattern and T_i a text substring.*

$$d_{\mathrm{H}}^{+}(i) \leqslant k \implies \mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k\,.$$

Proof. Let P and T_i be such that $d_{\mathrm{H}}^{+}(i) \leqslant k$. By definition there exists an α such that the set $J = \{\, j \mid P[j] + \alpha \neq T_i[j] \,\}$ has size at most k. As π is a permutation, there are at most $2k$ positions $j \in [m]$ such that either $j \in J$ or $\pi(j) \in J$. Therefore, for all (at least) $m - 2k$ remaining positions $j' \in [m]$ we have that $P[j'] + \alpha = T_i[j']$ and $P[\pi(j')] + \alpha = T_i[\pi(j')]$. For each such position j', by rearranging the two equations it follows from Lemma 27 that $P_\pi[j'] = (T_i)_\pi[j']$. Thus, there are at most $2k$ mismatches between P_π and $(T_i)_\pi$. □

A logical next step would be to attempt to find a permutation π with the property that $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k$ implies that $d_{\mathrm{H}}^{+}(i) \leqslant k$ for all P, T_i. Unfortunately, Lemma 29 below shows that no such permutation can exist. As Corollary 30 states, this immediately implies that there is no permutation which is k-tight for all P, T_i. Instead we will select our permutation at random and show that we can obtain a permutation that is k-tight for a given P, T_i with constant probability.

Lemma 29. *Let π be any permutation and $6 \leqslant k < m/4$. There exists a pattern P and text substring T_i such that*

$$d_{\mathrm{H}}^{+}(i) > k \quad \textit{and} \quad \mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k\,.$$

Proof. We define P to be an m-length string of zeros. In order to define the m-length string T_i we first introduce some notation.

Let $k' = \lfloor k/2 \rfloor + 1$. We identify a set of k' locations $\ell_0, \ell_1, \ldots, \ell_{k'-1} \in [m]$ as follows. Location $\ell_0 = 0$. For $q \in \{1, \ldots, k'-1\}$, location ℓ_q is the smallest position in $[m]$ that is not any of the preceding locations $\ell_0, \ldots, \ell_{q-1}$ or any location that is mapped to or from by any of these preceding locations (under π). Formally, ℓ_q is the smallest location which is not in the set $L_q = \left\{\, \ell_{q'}, \pi(\ell_{q'}), \pi^{-1}(\ell_{q'}) \;\middle|\; q' \in [q] \,\right\}$. Observe that the set L_q has size at most $3k' \leqslant 3(k/2+1) < 3k < m$ (since $k < m/4$), hence such a location always exists.

We can now define T_i as follows. For all $q \in [k']$, let $T_i[\ell_q] = 1$ and $T_i[\pi(\ell_q)] = 1$. At all other locations j, $T_i[j] = 0$. Observe that by construction, the locations $\ell_0, \ldots, \ell_{k'-1}$ and $\pi(\ell_0), \ldots, \pi(\ell_{k'-1})$ are all distinct. Therefore, T_i contains exactly $2k'$ ones and $m - 2k'$ zeros. As $2k' \leqslant k + 2 < m/2$, more than half the locations have $T_i[j] = P[j] = 0$, and therefore $d_{\mathrm{H}}^{+}(i)$ is minimised by the shift $\alpha = 0$. Thus, $d_{\mathrm{H}}^{+}(i) = 2k' > k$.

We proceed by showing that the alignment of P_π and $(T_i)_\pi$ contains at least $m - 3k'$ matches. There are $m - 2k'$ locations j in T_i such that $T_i[j] = 0$. Of these locations, at most $2k'$ have $T_i[\pi(j)] = 1$. Therefore, there are at least $m - 4k'$ locations j such that $T_i[j] = T_i[\pi(j)] = 0$. As $P[j] = P[\pi(j)] = 0$, we have by Lemma 27 that $P_\pi[j] = (T_i)_\pi[j]$ at $m - 4k'$ locations. Now consider locations ℓ_q for $q \in [k']$. By construction, $T_i[\ell_q] = T_i[\pi(\ell_q)] = 1$ and therefore $P_\pi[\ell_q] = (T_i)_\pi[\ell_q]$ by Lemma 27. This implies a further k' matching locations. There are therefore at least at least $m - 3k'$ matches or at most $3k'$ mismatches between P_π and $(T_i)_\pi$. Since $3k' \leqslant 3(k/2+1) \leqslant 2k$ for all $k \geqslant 6$ we have that $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k$. □

Corollary 30. *Let π be any permutation and $6 \leqslant k < m/4$. There exists a pattern P and text substring T_i for which π is not k-tight.*

Proof. Immediate from Definition 26 and Lemma 29. □

4.3.1 Random permutations

We will choose a permutation uniformly at random from a simple family of permutations. On first inspection, we could have chosen from the family of all permutations. We claim without proof that a permutation chosen uniformly at random from the family of all permutations is k-tight for any P, T_i with constant probability. However, we must be able to efficiently compute $\mathrm{Ham}(P_\pi, (T_i)_\pi)$ for all i under our chosen permutation. The key problem being that in general $(T_i)_\pi$ is not easily obtained from T. As i varies, $(T_i)_\pi$ could change drastically, even when i is only incremented by one. Therefore we must be careful in selecting our family of permutations.

We will use the family of cyclic permutations, denoted $\mathbb{C}_m$ (for patterns of length m), defined as follows.

Definition 31. The set $\mathbb{C}_m$ contains the $m-1$ cyclic permutations $\pi_1, \pi_2, \ldots, \pi_{m-1}$, where

$$\pi_q(j) = j + q \bmod m\,.$$

We now show in Lemma 32 that $\mathbb{C}_m$ has the desired property of k-tightness when $m > 6k^2$. There is a corner case when $k \in \{0, 1\}$ which is easily solved in $O(n)$ time using our deterministic algorithm from Section 4.2. For Lemma 32 we require that $k \geqslant 2$.

Lemma 32. *Let P be a pattern and T_i a text substring. When $m > 6k^2$ and $k \geqslant 2$,*

$$\frac{\left|\,\{\,\pi \mid \pi \in \mathbb{C}_m \text{ is } k\text{-tight for } P, T_i\,\}\,\right|}{|\mathbb{C}_m|} \geqslant \frac{1}{6}\,.$$

Proof. Let $\rho = \left|\,\{\,\pi \mid \pi \in \mathbb{C}_m \text{ is } k\text{-tight for } P, T_i\,\}\,\right| / |\mathbb{C}_m|$. We will show that $\rho \geqslant 1/6$. Note that $|\mathbb{C}_m| = m-1$. We let $h = d_{\mathrm{H}}^{+}(i)$ be the minimal number of mismatches between P and T_i, and $\widehat{\alpha}$ be the shift which minimises $d_{\mathrm{H}}^{+}(i)$.

Assume first that $h \leqslant k$. By Lemma 28 and Definition 26 we have that that every $\pi \in \mathbb{C}_m$ is k-tight for P, T_i and therefore $\rho = 1$. Assume second that that $h > k$. We split the proof into three cases:

$$\textbf{Case 1.}\quad k < h \leqslant 2k \qquad \textbf{Case 2.}\quad 2k < h \leqslant \frac{m}{3} \qquad \textbf{Case 3.}\quad \frac{m}{3} < h$$

First we introduce some notation. There are exactly $m-h$ positions j where $\widehat{\alpha}+P[j] = T_i[j]$. We call such a position an $\widehat{\alpha}$-match. Similarly, any position with $\alpha+P[j] = T_i[j]$ for some α is called an α-match. Positions which are not α-matches are called α-mismatches. Hence there are h distinct $\widehat{\alpha}$-mismatches. We will refer to $\pi(j)$ as the position that j is mapped to (by π).

Case 1 ($k < h \leqslant 2k$). Let j be an arbitrary $\widehat{\alpha}$-mismatch. Position j is mapped to another $\widehat{\alpha}$-mismatch in exactly $h-1$ distinct permutations of $\mathbb{C}_m$. This holds for each of the h distinct $\widehat{\alpha}$-mismatches. Hence there are at most $(h-1)h$ permutations under which some $\widehat{\alpha}$-mismatch is mapped to another $\widehat{\alpha}$-mismatch. The remaining (at least) $(m-1)-(h-1)h$ permutations π in $\mathbb{C}_m$ immediately have the following two properties:

(i) if position j is an $\widehat{\alpha}$-mismatch then $\pi(j)$ is an $\widehat{\alpha}$-match;

(ii) if position $\pi(j)$ is an $\widehat{\alpha}$-mismatch then position j is an $\widehat{\alpha}$-match.

There are h positions j with property (i) and another (disjoint) h positions j with property (ii). That is, for each $\widehat{\alpha}$-mismatch there are two positions j that meet one of the two properties above. By Lemma 27, each such j implies that $P_\pi[j] \neq (T_i)_\pi[j]$. Therefore, in each of these $(m-1)-(h-1)h$ permutations π, $\mathrm{Ham}(P_\pi, (T_i)_\pi) \geqslant 2h > 2k$ and so each such permutation is k-tight for P, T_i. By the assumption of Case 1, $h \leqslant 2k$, and the assumptions that $m \geqslant 6k^2$ and $k \geqslant 2$, we have that $(m-1)-(h-1)h \geqslant m-4k^2 \geqslant m/3$. Thus, $\rho \geqslant (m/3)/(m-1) > 1/6$.

Case 2 ($2k < h \leqslant m/3$). Let K be an arbitrary set of $2k$ distinct $\widehat{\alpha}$-mismatches. For any permutation π, let

$$K_\pi^{-1} = \{\, j \mid \pi(j) \in K \,\} .$$

We define

$$\mathrm{Ham}_K(P_\pi, (T_i)_\pi) = \left|\, \left\{\, j \;\middle|\; j \in (K \cup K_\pi^{-1}) \wedge P_\pi[j] \neq (T_i)_\pi[j] \,\right\} \right|$$

to be the number of mismatch positions between P_π and $(T_i)_\pi$ that are also in K or K_π^{-1}. We now consider the total number of mismatches between P_π and $(T_i)_\pi$ (that are in K or K_π^{-1}) summed over all permutations in $\mathbb{C}_m$. Let

$$H_K(P, T_i) = \sum_{\pi \in \mathbb{C}_m} \mathrm{Ham}_K(P_\pi, (T_i)_\pi) .$$

Since $h \leqslant m/3$ by the assumption of Case 2, there are at least $2m/3$ $\widehat{\alpha}$-matches. A permutation π that maps a position $j \in K$ to an $\widehat{\alpha}$-match creates a mismatch $P_\pi[j] \neq (T_i)_\pi[j]$ by Lemma 27 (as j is an $\widehat{\alpha}$-mismatch). For a fixed $j \in K$, the number of permutations in $\mathbb{C}_m$ that map j to an $\widehat{\alpha}$-match equals the number of $\widehat{\alpha}$-matches, which is at least $2m/3$. Thus, the set K of $2k$ $\widehat{\alpha}$-mismatches contributes at least $2k \cdot (2m/3)$ to $H_K(P, T_i)$.

Similarly, any position j which is an $\widehat{\alpha}$-match creates a mismatch $P_\pi[j] \neq (T_i)_\pi[j]$ by Lemma 27 if it is mapped to an $\widehat{\alpha}$-mismatch in K. This occurs under exactly $2k$ permutations. Recall that any j which is mapped to a position in K under π belongs to K_π^{-1}. Therefore, given that there are at least $2m/3$ $\widehat{\alpha}$-matches, the contribution is at least $2k \cdot (2m/3)$ further distinct mismatches to $H_K(P, T_i)$.

Summing up the previous two paragraphs, we have shown that $H_K(P, T_i) \geqslant (8/3)mk$. Each permutation π that is not k-tight for P, T_i has $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k$ (since $h > k$). Therefore, $m \cdot 2k$ is a generous upper bound on the number of mismatches across all permutations which are not k-tight. This leaves at least $(8/3)mk - 2mk = (2/3)mk$ mismatches among the k-tight permutations of $\mathbb{C}_m$. Since $|K \cup K_\pi^{-1}| \leqslant 4k$, we have that $\mathrm{Ham}_K(P_\pi, (T_i)_\pi) \leqslant 4k$ for any π, hence each permutation contributes at most $4k$ mismatches to $H_K(P, T_i)$. Therefore there are at least $(2/3)mk/(4k) = m/6$ distinct k-tight permutations. Thus, $\rho \geqslant (m/6)/(m-1) \geqslant 1/6$.

Case 3 ($m/3 < h$). Similarly to Case 2, we consider the total number of mismatches between P_π and $(T_i)_\pi$ summed over all permutations in $\mathbb{C}_m$. Let

$$H(P, T_i) = \sum_{\pi \in \mathbb{C}_m} \mathrm{Ham}(P_\pi, (T_i)_\pi)\,.$$

Since $h > m/3$ the number of α-mismatches is more than $m/3$ for all α. Fix an arbitrary position j and choose an α such that j is an α-match. There are at least $m/3$ permutations π in $\mathbb{C}_m$ that map position j to an α-mismatch. By Lemma 27, $P_\pi[j] \neq (T_i)_\pi[j]$ for each of these permutations. Hence position j will contribute with at least $m/3$ to $H(P, T_i)$. By considering all m positions j, we have that $H(P, T_i) \geqslant m \cdot (m/3)$.

Similarly to the reasoning in Case 2, each permutation π that is not k-tight for P, T_i has $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k$ (since $h > k$). Again, $m \cdot 2k$ is a generous upper bound on the number of mismatches across all permutations which are not k-tight. This leaves at least $m^2/3 - 2mk$ mismatches among the k-tight permutations of $\mathbb{C}_m$. As certainly $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant m$, we have that there are at least $m/3 - 2k$ distinct k-tight permutations for P, T_i. Therefore,

$$\rho \geqslant \frac{m/3 - 2k}{m - 1} \geqslant \frac{k - 1}{3k} \geqslant \frac{1}{6}\,,$$

where the second inequality follows from $m > 6k^2$ and the last inequality from $k \geqslant 2$, both assumptions in the statement of the lemma. □

4.3.2 The algorithm

Before describing the randomised algorithm we turn our attention to the problem of finding all positions $i \in [n - m + 1]$ such that $\mathrm{Ham}(P_\pi, (T_i)_\pi) \leqslant 2k$ under an arbitrary cyclic permutation $\pi \in \mathbb{C}_m$. We will describe a simple deterministic algorithm that computes $\mathrm{Ham}(P_\pi, (T_i)_\pi)$ by reduction to the conventional k-mismatch problem.

Let $\pi_q \in \mathbb{C}_m$ be a fixed but arbitrary permutation ($q \in [1, \ldots, m-1]$). Recall that $\pi_q(j) = j + q \bmod m$. We define

$$\begin{aligned} P_q^+ &= P_{\pi_q}[0 \ldots (m - q - 1)]\,, \\ P_q^- &= P_{\pi_q}[(m - q) \ldots (m - 1)]\,. \end{aligned}$$

Thus, $P_{\pi_q} = P_q^+ \| P_q^-$. We have $|P_q^+| = m - q$ and $|P_q^-| = q$. Now define T_q^+ and T_q^- such that

$$\begin{aligned} T_q^+[j] &= T[j + q] - T[j]\,, \\ T_q^-[j] &= T[j + q - m] - T[j]\,, \end{aligned}$$

for all $j \in [n]$ (except those that take the indices "out of range"). Observe that

$$(T_i)_{\pi_q} = T_q^+[i \ldots (i + m - q - 1)] \parallel T_q^-[(i + m - q) \ldots (i + m - 1)]\,,$$

Algorithm 4 Overview of randomised solution to SHIFT-k-DECISION.

1. Pick a cyclic permutation $\pi_q \in \mathbb{C}_m$ uniformly at random.
2. Construct the strings P_q^+, P_q^-, T_q^+ and T_q^-.
3. Run a $2k$-mismatch algorithm on the pairs (P_q^+, T_q^+) and (P_q^-, T_q^-) as a black box.
4. Using the results from Step 3 and Equation (5), compute $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q})$ for all i.
5. Any alignment i with $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) \leqslant 2k$ is declared to have $d_{\mathrm{H}}^+(i) \leqslant k$.

where the first substring has length $m-q$ and the second substring has length q. From these definitions it now follows directly that

$$\begin{aligned}\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) = \quad & \mathrm{Ham}\big(P_q^+, T_q^+[i \ldots (i+m-q-1)]\big) \\ & + \mathrm{Ham}\big(P_q^-, T_q^-[(i+m-q) \ldots (i+m-1)]\big)\,.\end{aligned} \tag{5}$$

Thus, in order to determine which positions i have $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) \leqslant 2k$, we first construct P_q^+, P_q^-, T_q^+ and T_q^-, and then we run a standard $2k$-mismatch algorithm on the pairs (P_q^+, T_q^+) and (P_q^-, T_q^-) and use the previous formula.

We can now finally give an overview of our randomised algorithm for the SHIFT-k-DECISION problem. The steps are described in Algorithm 4. The overall running time and proof of correctness is given in Theorem 33 below. The algorithm makes one-sided errors and outputs a false match (incorrectly reports $d_{\mathrm{H}}^+(i) \leqslant k$) with constant probability per alignment. As we will see in the proof of Theorem 33, by running the algorithm a logarithmic number of times drastically reduces the probability of an error occurring at one or more alignments.

Theorem 33. *For any choice of constant c,* SHIFT-k-DECISION *can be solved randomised in $O(cn\sqrt{k \log k} \log n)$ (deterministic) time when $k < \sqrt{m/6}$. The algorithm makes only false-positive errors (incorrectly declares the Hamming distance is at most k). With probability at least $1 - 1/n^c$, the algorithm is correct at every alignment.*

Proof. As discussed in Section 4.3.1, if $k \in \{0, 1\}$ then we can use the deterministic algorithm from Section 4.2 and achieve time complexity of $O(n)$ and no errors. Therefore, we focus on the case that $k \geqslant 2$.

We first consider correctness. It follows from the discussion above that Algorithm 4 does indeed determine, for every alignment i, whether $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) \leqslant 2k$. We first show that

(i) $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) \leqslant 2k$ when $d_{\mathrm{H}}^+(i) \leqslant k$;

(ii) the probability that $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) \leqslant 2k$ when $d_{\mathrm{H}}^+(i) > k$ is at most $5/6$.

By Lemma 28 we have that if $d_{\mathrm{H}}^+(i) \leqslant k$ then $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) \leqslant 2k$. This proves property (i). By Definition 26, $\mathrm{Ham}(P_{\pi_q}, (T_i)_{\pi_q}) > 2k$ if $d_{\mathrm{H}}^+(i) > k$ for all permutations

π_q that are k-tight for P, T_i. The permutation π_q is selected uniformly at random from $\mathbb{C}_m$ in Step 1, hence by Lemma 32 it is k-tight for P, T_i with probability at least $1/6$. This proves property (ii). Note that we can apply Lemma 32 since we have assumed that $m > 6k^2$ and $k \geqslant 2$.

As Algorithm 4 only makes false-positive errors, we can amplify the probability of giving correct outputs by repeating the algorithm. We repeat it $4(c+1)\lceil \log n \rceil$ times, where c is a constant, and output any alignment which is reported by all repeats. More precisely, let i be some alignment such that $d_{\mathrm{H}}^{+}(i) > k$. The probability that one run of Algorithm 4 incorrectly reports position i as a match is at most $5/6$. Thus, the probability that all runs output i as a match is at most

$$(5/6)^{4(c+1)\lceil \log n \rceil} < (1/2)^{(c+1)\log n} < 1/n^{c+1}\,.$$

By the union bound over all positions i, the probability of the multi-run algorithm outputting a false match in at least one alignment is at most $n \cdot 1/n^{c+1} = 1/n^c$ as required.

We now consider the time complexity of Algorithm 4 (without amplification). Step 1 requires only constant time to pick a permutation at random. Step 2 requires $O(n)$ time by inspection of the definitions. Step 3 makes two calls to a $2k$-mismatch algorithm. For both calls the input is a pattern of length $O(m)$ and a text of length $O(n)$. Using the fastest known k-mismatch algorithm of Amir et al. [ALP04], this step takes $O(n\sqrt{k \log k})$ time. Steps 4 and 5 require only scanning the output of Step 3 and therefore take $O(n)$ time. This gives a time complexity of $O(n\sqrt{k \log k})$ time. However, we repeat the algorithm $O(c \log n)$ times to reduce the error probability, hence $O(cn\sqrt{k \log k} \log n)$ is the total time complexity. □

5 Discussion

We have shown how to derive both new upper and lower bounds for a variety of pattern matching problems under polynomial transformations. In some cases we have improved on known results and in others introduced new problem definitions and solutions. There remain however a number of open questions. First, we suspect that the true complexity of Poly-r-$L_2^\star$ is unresolved, particularly for higher polynomial transformations. For example, when $r = m$ there exists a straightforward $O(nm)$ time solution by considering the problem independently at each alignment. It is also still uncertain if the normalised Hamming distance problem is 3Sum-hard for polynomials of degree greater than one. For Shift-k-Decision, our fast randomised algorithm applies only when $k < \sqrt{m/6}$. However, our lower bound for the same problem applies to the case where we want to determine if the Hamming distance is at most $m - 2$. This leaves a range of values of k where the complexity is not yet determined. It is also an interesting question whether our randomised solution can be efficiently modified to output the Hamming distance at each alignment rather than simply a decision about whether it is greater or less than k or indeed if a new fast method can be found for this problem which will allow the presence of wildcards in the input.

6 Acknowledgements

MJ and BS are both supported by the EPSRC. The authors are grateful to Philip Bille for very helpful discussions on the topic of 3Sum and related problems and their application to lower bounds for pattern matching problems.

References

[AALP06] A. Amir, Y. Aumann, M. Lewenstein, and E. Porat. "Function matching". *SIAM Journal on Computing* 35.5 (2006), pp. 1007–1022.

[Abr87] K. Abrahamson. "Generalized string matching". *SIAM Journal on Computing* 16.6 (1987), pp. 1039–1051.

[AF95] A. Amir and M. Farach. "Efficient 2-dimensional Approximate Matching of Half-rectangular Figures". *Information and Computation* 118.1 (1995), pp. 1–11.

[ALP04] A. Amir, M. Lewenstein, and E. Porat. "Faster Algorithms for String Matching with k Mismatches". *Journal of Algorithms.* 50.2 (2004), pp. 257–275.

[ALPU05] A. Amir, O. Lipsky, E. Porat, and J. Umanski. "Approximate Matching in the L_1 Metric". *CPM '05: Proc. 16th Annual Symp. on Combinatorial Pattern Matching*, pp. 91–103.

[Ata01] M. J. Atallah. "Faster image template matching in the sum of the absolute value of differences measure." *IEEE Transactions on Image Processing* 10.4 (2001), pp. 659–663.

[BDP05] I. Baran, E. D. Demaine, and M. Pătraşcu. "Subquadratic Algorithms for 3SUM". *WADS '05: Proc. 9th Workshop on Algorithms and Data Structures*, pp. 409–421.

[CC07] P. Clifford and R. Clifford. "Simple Deterministic Wildcard Matching". *Information Processing Letters* 101.2 (2007), pp. 53–54.

[CCI05] P. Clifford, R. Clifford, and C. Iliopoulos. "Faster Algorithms for δ,γ-Matching and Related Problems". *CPM '05: Proc. 16th Annual Symp. on Combinatorial Pattern Matching*, pp. 68–78.

[CH02] R. Cole and R. Hariharan. "Verifying candidate matches in sparse and wildcard matching". *STOC '02: Proc. 34th Annual ACM Symp. Theory of Computing*, pp. 592–601.

[CI04] R. Clifford and C. Iliopoulos. "String Algorithms in Music Analysis". *Soft Computing* 8.9 (2004), pp. 597–603.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[CW90] D. Coppersmith and S. Winograd. "Matrix multiplication via arithmetic progressions". *Journal of Symbolic Computation* 9.3 (1990), pp. 251–280.

[Eri99a] J. Erickson. "Lower Bounds for linear satisfiability problems". *Chicago Journal of Theoretical Computer Science* 8 (1999).

[Eri99b] J. Erickson. "New Lower Bounds for Convex Hull Problems in Odd Dimensions". *SIAM Journal on Computing* 28 (1999), pp. 1198–1214.

[ES95] J. Erickson and R. Seidel. "Better lower bounds on detecting affine and spherical degeneracies". *Discrete and Comp. Geometry* 13.1 (1995), pp. 41–57.

[GO95] A. Gajentaan and M. H. Overmars. "On a class of $O(n^2)$ problems in computational geometry". *Computational Geometry* 5.3 (1995), pp. 165–185.

[Kin04] J. King. *A Survey of 3*SUM*-Hard Problems.* . Dec. 2004. URL: `http://www.cs.mcgill.ca/~jking/papers/3sumhard.pdf`.

[Kos87] S. R. Kosaraju. "Efficient string matching". Manuscript. 1987.

[Lew95] J. P. Lewis. "Fast template matching". *Vision Interface '95*, pp. 120–123.

[LP05] O. Lipsky and E. Porat. "Approximate Matching in the L_∞ Metric". *SPIRE '05: Proc. 12th International Symp. on String Processing and Information Retrieval*, pp. 331–334.

[LU00] K. Lemström and E. Ukkonen. "Including interval encoding into edit distance based music comparison and retrieval". *AISB '00: Proc. Society for the Study of Artificial Intelligence and the Simulation of Behaviour*, pp. 53–60.

[LV86] G. M. Landau and U. Vishkin. "Efficient string matching with k mismatches". *Theoretical Computer Science* 43 (1986), pp. 239–249.

[MNU05] V. Mäkinen, G. Navarro, and E. Ukkonen. "Transposition Invariant String Matching". *Journal of Algorithms.* 56.2 (2005), pp. 124–153.